Fast and Accurate NoC Latency Estimation for Application-Specific Traffics via Machine Learning

Yang Li and Pingqiang Zhou

***Abstract*—Latency is one of the critical performance metrics for Networks-on-Chips (NoCs). When designing an NoC, the designers have to explore enormous design parameters and various traffic patterns, thus a fast and accurate latency estimator is essential to explore the large design space. In this brief, we present an ideal neural network-based methodology for latency estimation in NoCs, especially for application-specific traffics. By inputting the sequence of extracted traffic features and NoC parameters, the neural network model will infer the corresponding average network latency in a fast while accurate way. Instead of training one neural network model for each benchmark from scratch, we adopt transfer learning to train the network model for a new benchmark from another trained one. Experimental results on a set of widely used application-specific NoC benchmarks show that, our method can achieve an average estimation accuracy of 95%, and a 17.1X speedup for large NoCs compared to BookSim2 simulations. Our method can also achieve 20% to 70% improvement in accuracy over the other state-of-art machine learning-based works.**

***Index Terms*—Network-on-chips (NoCs), latency estimation, neural networks, application-specific traffic.**

I. Introduction

NETWORKS-ON-CHIP (NoCs) has become the mainstream communication solution for modern multi-core systems. An NoC designer faces at least two key challenges. First, there are a large group of design parameters to tune [1], such as topology, buffer depth, virtual channel and etc. Exploring the design space and finding an optimal solution is an NP-hard problem, facing great challenges [2]. It is essential to develop ideal evaluation methodologies for key performance metrics to accelerate design exploration. Second, when engineers design a new NoC, it's tedious to go through the design process from scratch. Thus, it is a meaningful but challenging task to re-use the valuable knowledge and experience from the previous design instances to guide the new design.

Network latency is one of the most important performance metrics for NoC design. Traditionally, cycle-accurate simulators are widely used to estimate the network latency in NoC design exploration, and famous examples include BookSim [3] and Noxim [4]. However, while the network size increases, the run-time per simulation grows up to hours, which is unacceptable for design exploration. Several researchers thus propose analytical estimation models for router delay, based on queue theories [5], [6], [7], the power-law distribution [8] and the generalized exponential distribution [9]. Although analytical methods are fast in evaluations, they rely on strong assumptions about the router architectures and traffic characterizations, which introduces inevitable errors. Besides, analytical models are not suitable for application-specific traffics since they are incompatible with the strict assumptions [2].

Recently, a few pioneering researchers have started to use machine learning (ML) for latency estimation in NoC design, including the Support-Vector Regression(SVR) [10], [11], the neural network [12], [13] and Graph Neural Network(GNN) [14]. To use ML methods, a designer has to collect a large amount of training samples by simulations, which takes a huge amount of time.

In our work, we propose to use neural network to build a fast while accurate latency estimator to aid NoC design exploration, especially targeting application-specific traffics. First, to reduce the data collection cost and improve the training efficiency, we adopt the transfer learning technique, which allows us to find a model for a new NoC design by reusing the trained model of another NoC, together with a much smaller set of new training data. Additionally, transfer leaning helps us to find a better model – the old trained model plays the role as a low-level feature extraction processor. With the new data, the transferred model learns more details of the input features, which assures higher estimation accuracy than the model trained from scratch [15]. Second, our work focuses on application-specific traffics. To the best knowledge of us, most prior works on NoC latency estimation only consider the synthetic traffic patterns, which are mathematically designed. Their features can be easily captured by either an analytical model or a machine learning model. In contrast, application-specific traffics are extracted from real-world application networks, whose characteristics are complicated to describe in a delicate way [1]. The comparisons among the prior related works and our work (NN+Transfer Learning) are illustrated in Table I.

The contributions of our work are summarized as follows:

- For the first time, we propose a transferable design flow for building an NoC latency estimator under different application-specific traffics. Our flow is applicable to a wide variety of communication scenarios.
- We present a general method to extract the traffic features of traffic flow and congestion. We then use the traffic

Manuscript received 17 November 2022; revised 10 February 2023; accepted 1 March 2023. Date of publication 17 March 2023; date of current version 29 August 2023. This work was supported by the National Natural Science Foundation of China under Grant 62074100. This brief was recommended by Associate Editor H. Yu. *(Corresponding author: Yang Li.)*

The authors are with the School of Information Science and Technology, ShanghaiTech University, Shanghai 201210, China (e-mail: liyang3@shanghaitech.edu.cn; zhoupq@shanghaitech.edu.cn).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TCSII.2023.3258700.

Digital Object Identifier 10.1109/TCSII.2023.3258700

TABLE I
COMPARISONS AMONG DIFFERENT ESTIMATION METHODS

Method	Simulation [3], [4]	Analytical [5]–[9]	ML [10]–[14], [16]	Our work NN+Transfer Learning
Runtime	Hours/days	Fast	Fast	Fast
Accuracy	High	W/ error	High	High
Data Cost	-	-	Large	Small

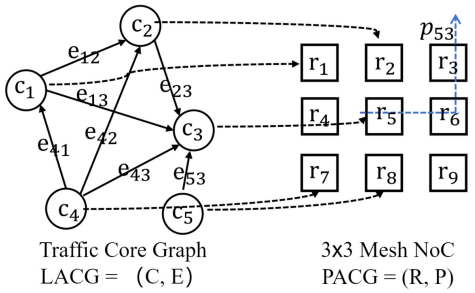

Fig. 1. One example of mapping from communication task graph to NoC hardware architecture.

features, together with varying NoCs design parameters, as the input of the latency estimator.

- We apply our design flow on five application-specific benchmarks. Results show that our method can achieve up to 17.1X speedup over the BookSim2 simulator [3], with estimation accuracy up to 98%. Compared with the state-of-art machine learning works, our proposed method can always improve the estimation accuracy of network latency by up to 70%.

II. PROBLEM FORMULATION

In our work, we assume the mapping from a given communication task graph, as presented by Logic Application Communication Graph (LACG), to the NoC architecture, as represented by a Physical Architecture Characterization Graph (PACG) [17], has been given as inputs to our problem formulation.

A LACG is a weighted directed graph G = (C, E), where $c_i \in C$ is a processing core that sends and receives messages while edge $e_{ij} \in E$ represents the traffic flow from c_i to c_j. A PACG is also a weighted directed graph G' = (R, P) which shows the hardware architecture of NoCs, where $r_i \in R$ is a router and $p_{ij} \in P$ represents the amount of transmission flow from r_i to r_j. Fig. 1 shows one example of mapping from LACG to PACG. In our work, we focus on application-specific traffics where the traffic patterns in LACG are irregular and the amount of communication demands among the processing cores are typically unbalanced.

After the mapping process, our work aims to find the average network latency of the mapped NoCs with some given set of design parameters (see Section III-A) in a fast while accurate way. The average network latency is measured as the weighted superposition of each traffic flow latency over the sum of communication flows. Mathematically, it can be stated as

$$L_{network} = \frac{1}{\sum p_{sd}} \times \sum p_{sd} L_{sd} \tag{1}$$

where p_{sd} is the transmission flow weight sending from source router r_s to destination router r_d and L_{sd} is the path latency.

As stated in Section I, it is impractical to build analytical models for application-specific traffics [2], and how to develop an efficient while general machine learning-based estimator for application-specific traffics is still an open problem.

III. METHODOLOGY

A. The Overall Design Flow

The overall design flow is demonstrated in Fig. 2, which consists of feature extraction, model training and validation, then transfer learning.

For the loss function of the model training, we adopt the following widely used metrics

- MSE (mean square error), defined as

$$MSE = \frac{1}{n} \sum (y - \hat{y}) \tag{2}$$

where y is the ground truth, $\hat{y}$ is the estimation latency by NN model, and n is the size of validation set.

- MAE (mean absolute error), defined as

$$MAE = \frac{1}{n} \sum |y - \hat{y}| \tag{3}$$

MAE (also known as L1 loss) has a better robustness to the irregular values while MSE will lead to a more even error distribution.

- MAPE (mean absolute percentage error), defined as

$$MAPE = \frac{100\%}{n} \sum \left| \frac{y - \hat{y}}{y} \right| \tag{4}$$

which is the error rate of estimation.

- R^2 (R-squared score), defined as

$$R^2 = 1 - \frac{\sum (y - \hat{y})^2}{\sum (y - y^2)^2} \tag{5}$$

which describes the degree of model training - when the model fits the ground truth well, the R^2 score is very close to 1.

B. Feature Extraction

As shown in Fig. 3, the application demand LACG graph (See Fig. 1) can also be denoted by a $N \times 3$ matrix, where N is the number of edges in LACG while 3 columns are the sending core c_i, receiving core c_j and flow weight e_{ij} respectively. By applying the CAM mapping [18], one core of the LACG will be mapped to one router of the PACG. In this step, a mesh of size $m \times m$ will be used, which is the smallest mesh that can hold all the cores in LACG. N is the number of traffic flows mapped to PACG, and 3 columns correspond to the sending router r_i, receiving router r_j and the transmission rate p_{ij} from r_i to r_j respectively. After mapping and routing, the input traffic rate λ_i of each source router can be obtained and we can build a $m \times m$ source input traffic matrix for NoCs.

We then use 2 layers of Principle Component Analysis (PCA) to extract the significant features and reduce the dimension of source input traffic matrix from $m \times m$ to 3×3. PCA is one of the mostly used feature extraction technique as it can keeps the most key information. In our experiments, we found that

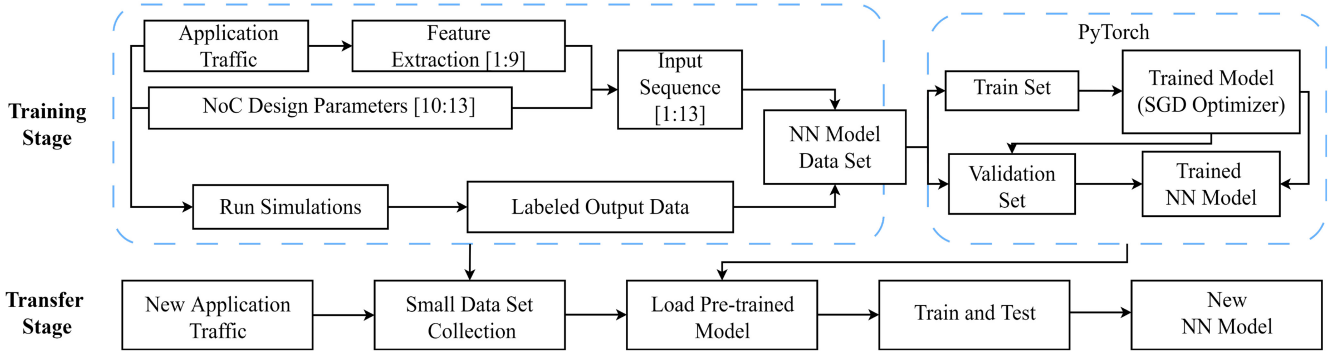

Fig. 2. The overall design flow of building Efficient NoCs latency estimation model for application-specific traffics.

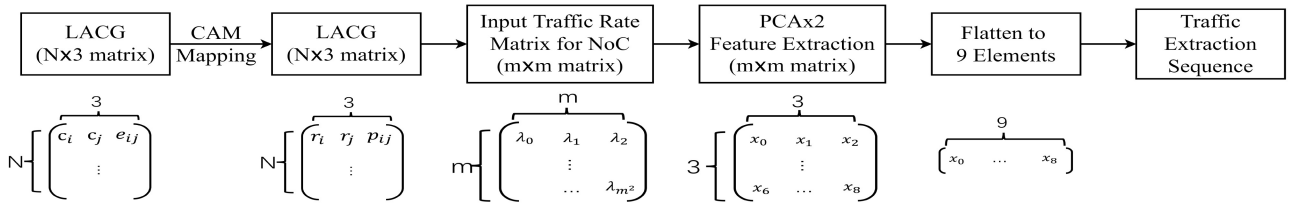

Fig. 3. Feature Extraction Procedure for Application-specific Traffic.

3×3 matrix can keep more than 99% of the key features for a wide spectrum of application-specific NoC benchmarks, from the 3×3 mesh to 15×15 mesh. Finally, we flatten the 3×3 PCA matrix into 9 elements as the traffic feature sequence.

C. Transfer Learning

To train a good NN model for network latency estimation, the recent related works [10], [11], [12], [13], [14], [16] require a large set of labeled data samples generated by simulations. This data collection step is most time-consuming and can take up to weeks.

In our work, we use transfer learning to greatly reduce the data collection cost, as shown in Fig. 4. The idea is as follows: For a small-scale NoC, we use the conventional training process armed with the feature extraction method discussed in Section III-B; then for the other larger NoCs designs, we apply the transfer learning technique, by reusing the trained model for the small NoCs design, together with a smaller set of data samples prepared for the new model. Training a model from scratch needs several hours but training models from a pre-trained one only need several minutes.

In addition to greatly reducing the data collecting cost and shortening the model training time, transfer learning can also capture the similarity in traffic features across different NoCs designs, and potentially improve the model accuracy.

IV. Experimental Results

Our experiments are conducted on a desktop with an i7-9700 core running at 3.4GHz, and the OS is Linux UBUNTU 18.04 LTS. All the data sets for training and validation are generated by BookSim2 [3]. All NN model processing steps are finished on PyTorch. The training optimizer is SGD with 0.9 momentum and 10^{-5} weight decay. To transfer the pre-trained model, we apply the original model weights and add new layers to train the new model.

The PACG in our work are mesh-based NoCs with wormhole switching and XY-routing. We choose five widely-used

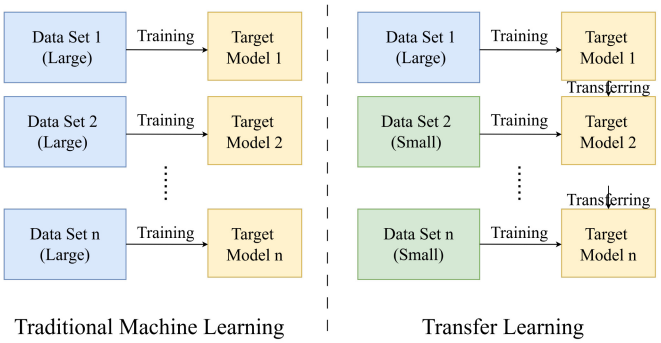

Fig. 4. Comparison between traditional machine learning and transfer learning for building the NN models for NoCs.

application benchmarks to test our approach – PIP, MWD, IMP, CMP and MPEG10 [19] whose details are demonstrated in Table III. The core number varies from 8 to 136 and the link number ranges from 8 to 294, which forms a big spectrum of NoCs design exploration space. The NN model structures of PIP, MWD and IMP are the same, with one input layer and three full-connected layers, and the number of neurons per layer ranges from 20 to 80. The ReLU and Sigmoid activation functions are used. The input elements details of the neural network are shown in Table II. While the transferred CMP and MPEG10 structures have the additional hidden layer compared with the pre-trained models. Besides, though we evaluate our method mainly on application-specific traffics, our work also use two synthetic traffic patterns, transpose, and tornado, to test the effectiveness of our work.

A. Run-Time Evaluation

Fig. 6 shows the comparison of run-time for the latency evaluation taken by our proposed model and BookSim2 simulator under the same application-traffic pattern whose details are shown in Table III. For small NoCs, using NN model approach only has a slightly cost reduction compared with simulation. But for those large NoCs, such as 9x9 or 12x12, our

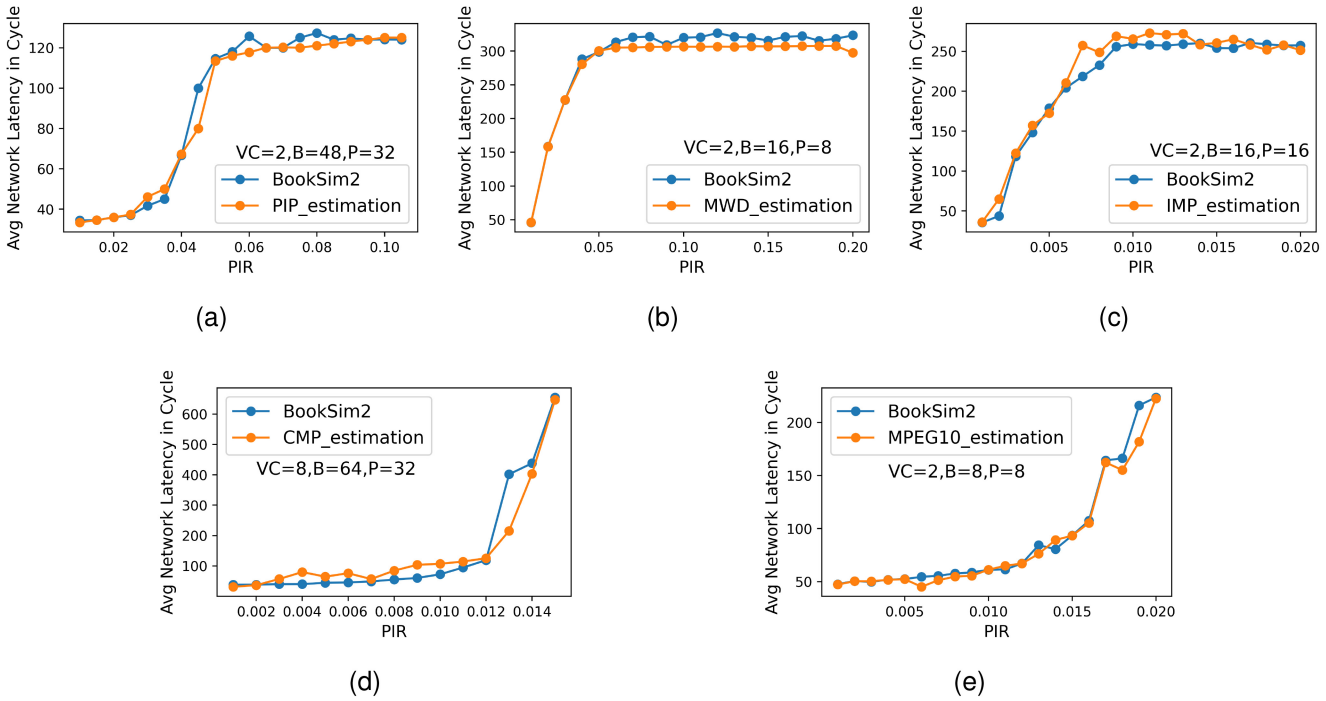

Fig. 5. The performance of our NN models under different NoC settings. (a) PIP, MAPE = 3.47% (b) MWD, MAPE = 3.3% (c) IMP, MAPE = 6% (d) CMP, MAPE = 9% (e) MPEG10, MAPE = 4.5%.

TABLE II
THE DETAILS OF INPUT ELEMENTS OF NN MODELS

Element	Input Parameters	Ranges
[1:9]	Traffic Extraction Matrix	ranges from 0 to 1
[10]	NoC Size	3x3, 4x4,...,12x12
[11]	Virtual Channel(VC)	1,2,...,10
[12]	Buffer Size(B)	8,16,...,64 flits
[13]	Packet Size(P)	8,12,...,32 flits

proposed method has a 12.77X and 17.1X speedup compared to the simulator. Thus, for those large NoCs, the proposed NN model method could greatly reduce the time cost of NoCs design space exploration. The run-time cost of NN model contains the training and inference time. Since the inference time is in milliseconds and extremely small compared with the training time, we only report the training time. Besides, in the real NoCs design process, designers always sweep NoCs parameters and adjust the injection rate to explore the design space. By imitating the same scenario, we also try 4 options of buffer size and 4 options of packet size. For the traffic injection rate, 50 adjustments in average are assumed for the successful design. Thus, 800 iterations should be simulated to find the best performance.

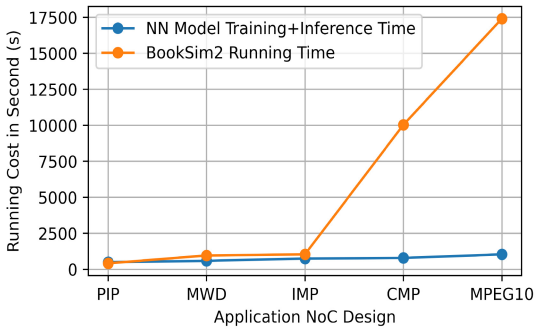

Fig. 6. Run-time vs. mesh size for simulations and proposed NN model.

TABLE III
THE PERFORMANCE OF OUR NN MODELS FOR FIVE APPLICATION-SPECIFIC BENCHMARKS

Benchmark	#Core	#Flow	Mesh Size	Accuracy
PIP	8	8	3x3	93%
MWD	12	13	4x4	96%
IMP	27	96	6x6	98%
CMP	69	136	9x9	91%
MPEG10	136	294	12x12	95%

B. Performance Analysis

We build one customized NN model for one NoC benchmark. First we build the NN model for the smallest benchmark PIP. Then we use transfer learning to obtain the models for the rest benchmarks. To be specific, the models of IMP and MWD are transferred from the PIP model, while CMP and MPEG10 models take IMP as the pre-trained model. It should be pointed out that if we use the conventional neural network training procedure, large data set with at least 5,000 samples should be collected per model to ensure the convergence of model. By using our transfer learning framework, small data set containing only 1,200 samples are needed, which vastly reduce the data collection cost for model training.

The performance results of our NN models are presented in Table III. The NN models for these benchmarks cover a large span of key NoC design parameters such as VC, P, B and PIR. Results show that our models achieve an average estimation accuracy of 91% to 98%, where the accuracy is calculated by [1-(MAPE of estimated latency)]x100%.

TABLE IV
COMPARISON OF OUR METHOD WITH OTHER ML METHODS ON APPLICATION-SPECIFIC TRAFFICS

ML	Linear Regression [13]			SVR-linear [10]–[12]			SVR-rbf [10]–[12]			Random Forest [16]			Proposed NN model		
Traffic	PIP	MWD	IMP	PIP	MWD	IMP	PIP	MWD	IMP	PIP	MWD	IMP	PIP	MWD	IMP
MAE	0.08	0.08	0.08	0.43	0.366	0.44	0.10	0.075	0.11	0.11	0.069	0.04	0.01	0.01	0.01
MAPE	65%	34%	80%	80%	2.191	1.45	21%	20%	47%	26.3%	14.6%	22.9%	7%	4%	9%
MSE	0.01	0.063	0.011	0.379	0.231	0.37	0.025	0.0137	0.026	0.07	0.011	0.002	0.00068	0.00039	0.00027
R2 socre	0.74	0.79	0.654	0.624	0.768	0.628	0.967	0.986	0.974	0.932	0.99	0.99	0.982	0.993	0.994

TABLE V
COMPARISON OF OUR METHOD WITH OTHER ML METHODS ON SYNTHETIC TRAFFICS

ML	Linear Regression [13]		SVR-linear [10]–[12]		SVR-rbf [10]–[12]		Random Forest [16]		Proposed NN model	
Traffic	Transpose	Tornado	Transpose	Tornado	Transpose	Tornado	Transpose	Tornado	Transpose	Tornado
MAE	0.08	0.01	0.34	0.0365	0.0608	0.035	0.108	0.008	0.006	0.005
MAPE	37.4%	1%	42%	11%	33.5%	6.3%	19.3%	11.7%	6%	0.6%
MSE	0.017	0.001	0.842	0.0036	0.021	0.0021	0.047	0.0079	0.00021	0.00054
R2 socre	0.86	0.99	0.215	0.99	0.979	0.99	0.95	0.99	0.994	0.998

We also present the model accuracy versus different PIRs for each benchmark in Fig. 5. Since our model considers a variety of NoC settings and contention situations, we also present the results with contentions for PIP, MWD and IMP. The latency versus PIR curve of each application benchmarks with the particular NoC setting is shown in Fig. 5. Results show that the latency estimated by our models matches closely to the results reported by the BookSim2 simulator. The MAPE of each estimator model is generally below 10% for these five benchmarks and most of them are under 5%. Additionally, the NN models are trained by MSE loss, leading to more uniform error rate distribution along the curves.

C. Comparisons With Other ML Methods

This section compares our NN models with the recently proposed machine learning works, including Linear Regression [13], SVR [10], [11], [12] and Random Forest [16]. We first test all these methods on PIP, MWD and IMP, and the results are shown in Table IV. It can be seen that the MAE and MSE of our proposed NN model are far below other ML methods. With regards to MAPE, other ML models range from 20% to 80%, some of the models even can not converge with an MAPE of up to 2.191. In contrast, our proposed NN model can greatly reduce the error rate – with MAPE only from 4% to 9% for the same traffic scenarios. As for the R2 score, our proposed model can converge very close to 1 in a short time.

Considering that prior ML works focus on synthetic traffic, we also test two synthetic traffics. Results presented in Table V show that, most models perform well under Tornado, but were just passable under Transpose. However, our proposed NN model can achieve an accuracy of 94% and 99.4%, as well as an R2 score of 0.994 and 0.998 respectively, which are superior to other methods.

V. CONCLUSION

In this brief, we present an ideal neural network-based methodology for fast and accurate latency estimation in NoCs, especially for application-specific traffics. We present a general method to extract the key features of traffic, then propose a transferable design flow for building ideal latency estimators across different NoCs designs. Experimental results on application benchmarks show that our method can achieve much more stable and better performance than prior related works.

REFERENCES

[1] W. J. Dally and B. P. Towles, *Principles and Practices of Interconnection Networks*. Amsterdam, The Netherlands: Elsevier, 2004.
[2] Z. Qian, P. Bogdan, C.-Y. Tsui, and R. Marculescu, "Performance evaluation of NoC-based multicore systems: From traffic analysis to NoC latency modeling," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 21, no. 3, pp. 1–38, 2016.
[3] N. Jiang et al., "A detailed and flexible cycle-accurate network-on-chip simulator," in *Proc. IEEE ISPASS*, 2013, pp. 86–96.
[4] V. Catania, A. Mineo, S. Monteleone, M. Palesi, and D. Patti, "Noxim: An open, extensible and cycle-accurate network on chip simulator," in *Proc. IEEE ASAP*, 2015, pp. 162–163.
[5] U. Y. Ogras and R. Marculescu, "Analytical router modeling for networks-on-chip performance analysis," in *Proc. DATE*, 2007, pp. 1–6.
[6] A. E. Kiasari, Z. Lu, and A. Jantsch, "An analytical latency model for networks-on-chip," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 21, no. 1, pp. 113–123, Jan. 2013.
[7] O. Matoussi, "NoC performance model for efficient network latency estimation," in *Proc. DATE*, 2021, pp. 994–999.
[8] P. Bogdan and R. Marculescu, "Statistical physics approaches for network-on-chip traffic characterization," in *Proc. IEEE CODES+ISSS*, 2009, pp. 461–470.
[9] Z. Qian, D.-C. Juan, P. Bogdan, C.-Y. Tsui, D. Marculescu, and R. Marculescu, "A comprehensive and accurate latency model for network-on-chip performance analysis," in *Proc. ASP-DAC*, 2014, pp. 323–328.
[10] Z. Qian, D.-C. Juan, P. Bogdan, C.-Y. Tsui, D. Marculescu, and R. Marculescu, "SVR-NoC: A performance analysis tool for network-on-chips using learning-based support vector regression model," in *Proc. DATE*, 2013, pp. 354–357.
[11] Z.-L. Qian, D.-C. Juan, P. Bogdan, C.-Y. Tsui, D. Marculescu, and R. Marculescu, "A support vector regression (SVR)-based latency model for network-on-chip (NoC) architectures," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 35, no. 3, pp. 471–484, Mar. 2016.
[12] A. Kumar and B. Talawar, "Machine learning based framework to predict performance evaluation of on-chip networks," in *Proc. ICCC*, 2018, pp. 1–6.
[13] B. Bhowmik, P. Hazarika, P. Kale, and S. Jain, "AI technology for NoC performance evaluation," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 68, no. 12, pp. 3483–3487, Dec. 2021.
[14] F. Li, Y. Wang, C. Liu, H. Li, and X. Li, "NoCeption: A fast PPA prediction framework for network-on-chips using graph neural network," in *Proc. DATE*, 2022, pp. 1035–1040.
[15] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Trans. Knowl. Data Eng.*, vol. 22, no. 10, pp. 1345–1359, Oct. 2010.
[16] J. Silva, M. Kreutz, M. Pereira, and M. Da Costa-Abreu, "An investigation of latency prediction for NoC-based communication architectures using machine learning techniques," *J. Supercomput.*, vol. 75, pp. 7573–7591, Aug. 2019.
[17] R. Marculescu, U. Y. Ogras, L.-S. Peh, N. E. Jerger, and Y. Hoskote, "Outstanding research problems in NoC design: System, microarchitecture, and circuit perspectives," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 28, no. 1, pp. 3–21, Jan. 2009.
[18] C.-L. Chou and R. Marculescu, "Contention-aware application mapping for network-on-chip communication architectures," in *Proc. IEEE ICCD*, 2008, pp. 164–169.
[19] S. Murali, *Designing Reliable and Efficient Networks on Chips*. Dordrecht, The Netherlands: Springer, 2009.